**Лабораторна робота №1. Розроблення плану тестування вимог до програмного забезпечення**

**Мета роботи**. Навчитися застосовувати на практиці знання щодо організації проведення тестування.

**Завдання на лабораторну роботу**

1. Ознайомитись з теоретичними відомостями, необхідними для виконання роботи.

2. На етапі створення специфікації вимог до програмного забезпечення системи потрібно проводити тестування вимог. Для організації цієї роботи для власного курсового проекту потрібно розробити план тестування вимог.

3. Оформити звіт.

4. Відповісти письмово на контрольні запитання.

**Зміст звіту**

1. Прізвище, назва групи, назва роботи.

2. Назва власного курсового проекту, мета роботи й завдання роботи, об’єкт і методи дослідження або розроблення.

3. План тестування за наступною схемою

| Вид вимоги | Роботи з тестування | Очікуваний результат |
| --- | --- | --- |
| Функціональні вимоги |  |  |
| Вимоги користувача |  |  |
| … |  |  |
|  |  |  |
| Нефункціональні вимоги |  |  |
| * обмеження, що витікають з предметної області |  |  |
| * вимоги до програмних інтерфейсів, надійності, обладнанню |  |  |
| * атрибути якості |  |  |
| * зовнішні системи та інтерфейси |  |  |
| * інші обмеження |  |  |

Для підготовки плану тестування потрібно заповнити таблицю

| Вид вимоги | Джерело вимоги | Вид документування |
| --- | --- | --- |
| Функціональні вимоги |  |  |
| Вимоги користувача |  |  |
| … |  |  |
|  |  |  |
| Нефункціональні вимоги |  |  |
| * обмеження, що витікають з предметної області |  |  |
| * вимоги до програмних інтерфейсів, надійності, обладнанню |  |  |
| * атрибути якості |  |  |
| * зовнішні системи та інтерфейси |  |  |
| * інші обмеження |  |  |

4. Висновки щодо цілей, які досягаються через проведення тестування .

**Контрольні запитання**

1. Через сукупність яких UML-діаграм Ви можете визначити вимоги (уточніть які саме)?
2. Для яких супроводжуючих документів до ПЗ важлива специфікація вимог?
3. Чому важливе тестування вимог?

Результати надсилати на електронну адресу викладача [**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com)у вигляді текстових файлів з іменем у форматі **OPI<Номер групи><Номер лекції / лабораторної>[-<Номер завдання>][літера позначення типу роботи L – лекція, R - лабораторна]<Прізвище англійською>**. Наприклад, **OPI4112R**buts.doc. Відповіді повинні бути не довгими і змістовними. Не копіюйте фрагментів з різних інформаційних джерел, подумайте і викладіть свою точку зору. При наявності відповідей-"близнюків" відповідь буде зараховуватися першому за часом надсилання.

**Строк відсилки ЛР ІПЗ-41 20.09.2021**

**ІПЗ-42 20.09.2021.**

**ТЕОРЕТИЧНІ ВІДОМОСТІ**

**Вимоги до програмного забезпечення** (ПЗ)— набір вимог щодо властивостей, якості та функцій ПЗ, що буде розроблено, або знаходиться у розробці, це специфікація того що повинно бути реалізовано. Це описи поведінки системи, властивості системи або її атрибути. Вони можуть бути обмеженні процесом розробки системи. [1] Вимоги визначаються в процесі аналізу вимог та фіксуються в специфікації вимог, діаграмах прецедентів та інших артефактах процесу аналізу та розробки вимог. Розробка вимог до програмної системи може бути розділена на декілька етапів:

* Знаходження вимог (збір, визначення потреб зацікавлених осіб та систем).
* Аналіз вимог (перевірка цілісності та закінченості).
* Специфікація (документування вимог).
* Тестування вимог.

Вимоги до ПЗ складаються з 3-х рівнів:

*бізнес-вимоги;*

*вимоги користувачів;*

*функціональні вимоги*.

При цьому кожна система характеризується *нефункціональними вимогами*. Модель на рис. 1 схематично показує організацію вимог. Овал – позначає тип інформації для вимог, прямокутник спосіб зберігання інформації (документи, діаграми, бази даних) [2].

**Бізнес-вимоги**(business requirements) містять високорівневі цілі організації або замовників ПЗ. Як правило, їх висловлюють, ті хто фінансує проект, покупці системи, менеджери користувачів, поясняючи для чого необхідна така система, визначено *цілі* які організація має намір досягнути використовуючи створене ПЗ.

**Вимоги користувачів** (user requirements) описують цілі і задачі, які дозволить користувачу вирішувати дана система, визначають набір завдань користувача, які повинна вирішувати програма, а також сценарії їхнього вирішення в системі. Ці вимоги можуть мати вигляд тверджень, варіантів використання, історій користувача, сценаріїв взаємодії..



Рис.1. Організація вимог до ПЗ

**Функціональні вимоги** (functional requirements) визначають функціональність ПЗ, яку розробники повинні забезпечити, щоб користувачі змогли виконати свої завдання в межах бізнес-вимог. Тобто визначають **«що»** повинен робити програмний продукт. Інколи їх називають вимоги поведінки (behavioral requirements), вони містять положення з традиційним «повинна». Наприклад, «Система повинна по електронній пошті відправляти користувачу підтвердження замовлення». Функціональні вимоги описують, що розробнику необхідно реалізувати.

Терміном **системні вимоги** (system requirements) позначають високорівневі вимоги до ПЗ програмної системи, яка містять багато підсистем.

**Бізнес-правила** (business rules) включають корпоративну політику, постанови управління, промислові стандарти і чисельні алгоритми. Бізнес-правила не є вимогами до ПЗ, тому що вони знаходяться зовні меж будь-якої системи ПЗ. Однак вони часто накладають обмеження, визначаючи хто може виконувати конкретні варіанти використання, або якими функціями повинна володіти система, яка підлягає відповідним правилам. Інколи бізнес-правила є джерелом атрибутів якості, які реалізуються в функціональності. Ці вимоги встановлюють обмеження, що витікають з предметної області

Функціональні вимоги документуються в **специфікації вимог до ПЗ** (software requirement specification, SRS), де описується очікувана поведінка системи. Це може бути документ, база даних, таблиця з вимогами, сховище даних в комерційному інструменті управління вимогами, або навіть набір карток для невеликого проекту. Специфікація вимог до ПЗ використовується при розробці, тестуванні, гарантії якості продукту, управлінні проектом і зв’язаних з проектом функціях.

Окрім функціональних в специфікації містяться **нефунціональні вимоги**, де описані цілі і атрибути якості.

**Атрибути якості (**quality attributes**)** є додатковим описом функцій продукту, опис характеристик важливих для користувачів або розробників. До таких характеристик відносяться легкість і простота використання, простота переміщення, цілісність, ефективність і стійкість до збоїв. Інші нефункціональні вимоги описують зовнішні взаємодії системи і середовища, а також обмеження дизайну і реалізації.

**Обмеження(constraints)** стосуються вибору можливості розробки зовнішнього вигляду і структури продукту.

**Характеристика** (feature) – це набір логічно зв’язаних функціональних вимог, які забезпечують можливості для користувача і відповідають бізнес-цілям.

**Характеристики окремих положень специфікації вимог**

*Повнота.* Кожна вимога повинна повно описувати функціональність, яку потрібно реалізувати в продукті, повинна містити, всю інформацію, необхідну для розробників, щоб їм вдалося створити цей фрагмент функціональності. Якщо вимога потребує уточнення, то використовують позначку «TBD» (to be determined - необхідно довизначити).

*Коректність.* Кожна вимога повинна точно описувати бажану функціональність. Для дотримання коректності необхідний зв'язок з джерелами вимог, наприклад з побажаннями користувачів або високорівневими системними. Вимоги до ПЗ, які конфліктують з «батьківськими» вимогами, не можна вважати коректними. Проте основна оцінка вимог - за представниками користувачів, тому їх необхідно надавати для перегляду.

*Здійсненність.* Необхідна можливість реалізувати кожну вимогу за відомих умов і обмежень системи і операційного середовища, визначення, що можна виконати технічно, а що немає можливості, або що зробити можна, але при додатковому фінансуванні.

*Необхідність.* Кожна вимога повинна відображати можливість, яка дійсно необхідна користувачам або яка потрібна для відповідності зовнішнім системним вимогам або стандартам. Відстежте кожну вимогу аж до стадії збору їх у користувачів, коли виявлялися варіанти використання, бізнес-правила або інші джерела.

*Визначення пріоритетів*. Визначте пріоритети кожної функціональної вимоги, характеристики або варіанту використання, щоб визначити, що необхідне для кожної версії продукту. Якщо всі положення однаково важливі, менеджерові проекту буде важко справитися із зменшенням бюджету, порушенням термінів, втратою персоналу або додаванням нових вимог в процесі розробки.

*Недвозначність.* Всі читачі вимог повинні інтерпретувати їх однаково. Документація повинна бути написана просто, коротко і точно, застосовуючи лексику, зрозумілу користувачам. «Ясність» — мета якості вимог, пов’язана з точністю. Всі спеціальні і заплутані терміни повинні бути внесені в спеціальний глосарій або словник.

*Можливість перевірки.* Потрібно розробити декілька тестів або використати інші прийоми для перевірки, наприклад експертизу або демонстрації, щоб встановити, чи дійсно в продукті реалізована кожна вимога. Порядок такої перевірки доцільно закласти при складанні специфікації вимог.

**Характеристики специфікації вимог**

Набір вимог, що становить специфікацію повинен відповідати таким характеристикам.

*Повнота.* Ніякі вимоги або необхідні дані не повинні бути пропущені.

*Узгодженість.* Узгоджені вимоги не конфліктують з іншими вимогами такого ж типу або з високорівневими призначеними для користувача, системними або бізнес-вимогами. Неузгодженість документів потрібно усунути до початку процесу розробки. Не завжди відомо, яке саме положення некоректне (якщо якесь некоректне), поки не виконано його дослідження. Рекомендується записувати автора кожної вимоги, щоб дізнатися, хто її висловив, якщо конфлікт все-таки буде виявлений.

*Здатність до модифікації.* Необхідно забезпечити можливість переробки вимог, якщо знадобиться, і підтримувати історію змін для кожного положення. Для цього всі вони повинні бути унікально помічені і позначені, щоб можна було посилатися на них однозначно. Кожна вимога повинна бути записана в специфікації тільки один раз, щоб не отримати неузгодженість, змінивши тільки одне положення з двох однакових. Доцільно використовувати посилання на первинні твердження, а не дублювати положення. Для спрощення модифікації специфікації потрібно скласти зміст документа і покажчик. Збереження специфікації в базі даних комерційного інструменту управління вимогами зробить їх придатними для повторного використання.

*Трасованість.* Трасованість, або можливість для аналізу, можна реалізувати як в напрямі назад, до першоджерел, так і вперед, до елементів дизайну і початкового коду, який його реалізує, а також до варіантів використання, які дозволяють перевірити коректність реалізації. Трасовані вимоги помічають відповідними ідентифікаторами. Вони записані в структурованій, деталізованій формі, в протилежність параграфам в довгій оповідній формі. Для виконання трасування потрібно уникати злипання безлічі вимог в одну.

**Джерела вимог.**

Основним джерелом вимог до інформаційної системи є міркування, висловлені представниками Замовника. У відповідність з ієрархічною моделлю вимог дана інформація структурується як мінімум на 2 рівні: *бізнес-вимоги* і *вимоги користувачів*. Оскільки вимоги формулюються до створюваної, ще не існуючої системи, тобто по суті вирішується завдання щодо проектування програмної системи, а представники Замовника далеко не завжди бувають компетентні в даному питанні, тому, разом з вимогами, висловленими Замовником, доцільно збирати і вимоги від інших співвласників системи: співробітників аналітичної групи виконавця, зовнішніх експертів і т. д. На вимоги співвласників зазвичай не накладається ніяких спеціальних обмежень.

Іншим важливим джерелом інформації, окрім виявлення вимог, є *артефакти*, що описують предметну область. Це можуть бути документи з описом бізнес-процесів підприємства, виконані консалтинговим агентством, або просто документи (посадові інструкції, розпорядження, зведення бізнес-правил) прийняті на підприємстві. Однією з небагатьох методологій, в якій спеціально виділяється робочий потік ділового моделювання, є Rational Unified Process (RUP).

**Атрибути якості ПЗ**

**Атрибути, важливі для користувачів**

**Доступність.** Під доступністю розуміється запланований *час доступності* (uptime), протягом якого система доступна для використання і повністю працездатна. Формально доступність дорівнює *середньому часу до збою* (mean time to failure, MT-TF) системи, діленому на суму середнього часу до збою і очікуваного часу до відновлення системи після збою. На доступність впливають періоди планового технічного обслуговування.

**Ефективність -** це показник того, наскільки ефективно система використовує продуктивність процесора, місце на диску, пам'ять або смугу пропускання з'єднання. Якщо система витрачає дуже багато доступних ресурсів, користувачі відмітять зниження продуктивності. Недостатня продуктивність дратує користувачів ставить під удар безпеку, наприклад, при перевантаженні системи контролю процесів реального часу. Потрібно визначити мінімальну конфігурацію устаткування, при якій вдається досягти заданих ефективності, пропускній спроможності і продуктивності. Щоб дозволити нижню межу у разі непередбачених умов і визначити подальше зростання, ви можете скористатися, наприклад, таким формулюванням: **"**Як мінімум 25% пропускної спроможності процесора і оперативної пам'яті, доступної застосуванню, не повинно використовуватися в умовах запланованого пікового навантаження." Типові користувачі не формулюють вимоги до ефективності, вони можуть задати час відгуку або заповнення простору на диску. Справа аналітика – поставити питання, які виявлять очікування користувачів про прийнятне зниження продуктивності, можливі піки навантаження і очікуване зростання.

**Гнучкість –** показує з якою легкістю в продукт вдається додати нові можливості. Якщо очікується, що при розробці доведеться вносити безліч поліпшень, варто вибрати такі рішення, які дозволять збільшити гнучкість ПЗ. Цей атрибут важливий для продуктів модель розробки, яких включає поліпшення і повтор успішних випусків, або розвиток прототипу.

***Вимоги до якості повинні записуватися у форматі, який передбачає можливість їх вимірювання.***

**Цілісність –** включає безпеку пов'язану з блокуванням неавторизованого доступу до системних функцій, запобіганням втраті інформації, антивірусним захистом ПЗ і захистом конфіденційності і безпеки даних, введених в систему. Цілісність дуже важлива для Інтернет-застосувань. Користувачі систем електронної комерції хочуть забезпечити дані своїх кредитних карток. Відвідувачі Web-сайтів не бажають, щоб приватна інформація про них або список відвідуваних ними сайтів використовувалися не за призначенням, а постачальники послуг доступу до Інтернету хочуть захиститися від атак типу «відмова в обслуговуванні» і інших хакерських атак.

У вимогах до цілісності немає місця помилкам. Потрібно використовувати такі точні терміни для формулювання вимог до цілісності: перевірка ідентифікації користувача, рівні привілеїв користувача, обмеження доступу або певні дані, які повинні бути захищені.

Потрібно уникати визначень вимоги до цілісності у вигляді обмежень дизайну, як, наприклад, вимоги до пароля для контролю доступу. Реальна вимога повинна обмежувати доступ до системи неавторизированих користувачів; паролі – всього лише один із способів (хоча і найпоширеніший) виконання цього завдання. Підхід заснований на вибраному методі ідентифікації користувачів як базової вимоги до цілісності вплине на певні функціональні вимоги, які реалізують функції аутентифікації в системі.

**Здатність до взаємодії** показує, яким чином система обмінюється даними або сервісами з іншими системами. Щоб оцінити здатність до взаємодії, необхідно знати, які застосування клієнти будуть використовувати спільно з розроблюваним продуктом і обмін яких даних передбачається. Як альтернативу можна визначити декілька функціональних вимог, що відносяться до операції імпорту.

**Надійність –** так називається вірогідність роботи ПЗ без збоїв протягом певного періоду часу. Іноді однією з характеристик надійності вважають стійкість до збоїв.

Вимірюють надійність ПЗ:

* як відсоток успішно завершених операцій;
* середній період часу роботи системи до збою.

Визначають кількісні вимоги до надійності, ґрунтуючись на тому, наскільки серйозними виявляться наслідки збоїв і чи виправдана ціна підвищення надійності. Системи, для яких потрібна висока надійність, потрібно проектувати з високим ступенем можливості тестування, щоб полегшити виявлення недоліків, що негативно впливають на надійність.

**Стійкість до збоїв (**відмовостійкість, fault tolerance) **–** тут розуміють рівень, до якого система продовжує коректно виконувати свої функції, не дивлячись на невірне введення даних, недоліки підключених програмних компонентів або компонентів устаткування або несподівані умови роботи. Стійке до збоїв ПЗ легко відновлюється після різних проблем і «не помічає» помилок користувачів. Потрібно опитати користувачів, які помилкові ситуації можливі при роботі з системою і як система повинна на них реагувати. Наприклад, якщо при роботі з редактором відбувся збій і користувач не встиг зберегти файл, то редактор повинен відновити всі зміни, внесені раніше, ніж за хвилину до збою, при наступному запуску програми даним користувачем.

**Атрибути, важливі для розробників**

**Легкість в експлуатації.** Цей атрибут показує, наскільки зручно виправляти помилки або модифікувати ПЗ. Легкість в експлуатації залежить від того, наскільки просто розібратися в роботі ПЗ, змінювати його і тестувати, і тісно пов'язано з гнучкістю і тестованістю. Цей показник украй важливий для продуктів, які часто змінюють, і тих, що створюються швидко (і, можливо, з економією на якості). Легкість в експлуатації вимірюють, використовуючи такі терміни, як *середній час потрібний для дозволу проблеми* і *відсоток коректних виправлень*. Наприклад, 1) вкладеність функцій, що викликаються, не повинна перевищувати два рівні; 2) для кожного програмного модуля непорожні коментарі в співвідношенні до початкового коду повинні складати як мінімум 0,5. Останнє завдання доцільно обговорювати з програмістами, що займаються технічним обслуговуванням, щоб зрозуміти, які якості початкового коду полегшать їм внесення змін або виправлення недоліків.

**Легкість переміщення.** Мірою її вимірювання можна вважати зусилля, необхідні для переміщення ПЗ з одного операційного середовища в іншу, а також можливість інтернаціоналізації і локалізації продукту. Важливо визначити ті частини продукту, які необхідно легко переміщати в інші середовища, і описати ці цільові середовища, щоб потім розробники вибрали способи розробки і кодування, які збільшать мобільність продукту.

**Можливість повторного використання.** Постійне завдання розробки ПЗ - можливість повторного використання - показує зусилля, необхідні для перетворення програмних компонентів; з метою їх подальшого застосування в інших застосуваннях. Витрати на розробку ПЗ з можливістю повторного використання значно вище, ніж на створення компоненту, який працюватиме тільки в одному застосуванні. Воно повинне бути модульним, добре задокументованим, не залежати від конкретних застосування і операційного середовища, а також володіти деякими універсальними можливостями. Цілі багатократного використання складно кількісно виміряти. Потрібно вказати, які елементи нової системи необхідно спроектувати так, щоб спростити їх повторне застосування, або вказати бібліотеки компонентів багатократного використання, які необхідно створити додатково до проекту.

**Тестованість.** Цей атрибут також називають *перевіряємість,* він показує легкість, з якою програмні компоненти або інтегрований продукт можна перевірити на предмет дефектів. Такий атрибут важливий для продукту, в якому використовуються складні алгоритми і логіка або є тонкі функціональні взаємозв'язки. Тестованість також важлива в тому випадку, якщо продукт необхідно часто модифікувати, оскільки передбачається піддавати його частому регресивному тестуванню, щоб з'ясувати, чи не погіршують внесені зміни існуючу функціональність. Наприклад, максимальна цикломатична складність модуля не повинна перевищувати 20.

*Цикломатична складність* (cyclomatic complexity) - це кількість логічних відгалужень в модулі початкового коду. Чим більше відгалужень і циклів в модулі, тим важче його тестувати, розуміти і підтримувати. Звичайно, проект не буде провалений. якщо значення цикломатичної складності одного з модулів досягне 24, проте ця директива вказала розробникам рівень якості, до якої слід прагнути. Якби її (вона представлена як вимога до якості) не було, не факт, що розробники при написанні своїх програм взяли б до уваги таку характеристику, як цикломатична складність. В результаті код програми міг стати таким заплутаним, що його ретельне тестування і доповнення виявилося б практично неможливим, а відладка взагалі стала б кошмаром.

**ТЕСТУВАННЯ ВИМОГ**

Вимоги - основа майбутньої системи

**Вимоги** це -

* функціональна характеристика системи, необхідна замовникові для того щоб вирішити проблему або досягти поставлених цілей;
* функціональна характеристика, якою повинна володіти система для того щоб бути відповідною контракту, стандарту, специфікації або іншому формальному документу;
* документ, що описує функціональні характеристики позначені вище.

Існують різні типи вимог:

**Бізнес вимоги -** цілі і завдання підприємства

**Вимоги користувачів -** потреби окремих (груп) замовників

**Функціональні вимоги -** описують **що** повинна робити система

**Нефункціональний вимоги -** описують **як** повинна працювати система

**Припущення і обмеження -** аспекти наочної області які обмежують або

впливають на дизайн системи

**Методи тестування вимог**

*Перед тестуванням вимог*

* Вимоги проаналізовані і задокументовані аналітиком
* Аналітик самостійно провів перевірку

*Під час тестування вимог*

* Зацікавлені особи підтверджують, що вимоги коректні і зрозуміли
* Користувачі підтверджують, що вимоги відображають їх потреби

*Критерій готовності вимог*

У вимогах міститься достатньо інформації для початку розробки

*Учасники тестування вимог:* Замовник, користувачі, проектна команда

*Тестування вимог грає важливу роль в процесі розробки ПЗ:*

* зменшення кількості доопрацювань і змін;
* скорочення ризиків;
* ознайомлення і узгодження завдань між розробниками.

*Методи тестування вимог*

* Перевірка документації
* Аналіз поведінки системи
* Прототипування

**Перевірка вимог**

Перевірка вимог найпоширеніший метод тестування вимог

*Опис методу:* Послідовний перегляд і перевірка всіх вимог

*Дозволяє перевірити на:* правильність, повноту, простежуваність, важливість, зрозумілість, однозначність і вимірність.

*Застосовується:* замовниками, аналітиками, проектними менеджерами, тестувальниками.

*Сильні сторони*

* Простота використання
* Відсутність спеціальних вимог до перевіряючого
* Покриває багато критеріїв якості
* Менші витрати часу

*Слабкі сторони*

* Якість перевірки залежить від перевіряючої
* Залучення різних фахівців
* Наявність документа з вимогами

**Аналіз поведінки системи**

*Опис методу*

Формування вимог у форматі вхід-вихід, подія- наслідок, умова-відповідь

*Застосовується найчастіше* Тестувальниками (test cases) та Аналітиками (use cases)

*Дозволяє перевірити на* повноту, зрозумілість, однозначність

*Сильні сторони*

* Добре перевіряє вимоги
* Представляє вимоги в структурованому і зрозумілому вигляді
* Результати методу легко використовуються для створення сценаріїв тестування

*Слабкі сторони*

* Вимагає більшої кількості часу
* Вимагає спеціальної підготовки

**Прототипування**

*Опис методу*

Створення моделі майбутньої системи

*Застосовується частіше для перевірки* повноти, правильності, реалізовуваності, зручності використання

*Використовується* аналітиками, архітекторами

*Приклад використання*

* Горизонтальний – модель якомога ширшого набору функціональності
* Вертикальний – докладна вузьконаправлена модель невеликої частини системи
* Throw-away – не використовується в подальшій розробці
* Еволюційний – поступово допрацьовується і перетворюється на готову систему

*Сильні сторони*

* Користувачі дістають можливість перевірити рішення
* Наочна допомога для розробників і тестувальників
* Перевірка вимоги на ту, що реалізовується і на зручність призначеного для користувача інтерфейсу

*Слабкі сторони*

* Вимагає значного часу
* Спеціальна підготовка для створення еволюційного прототипу
* Вимоги важливо тестувати для того що б гарантувати високу якість системи
* У вимог є багато критеріїв, яким вони повинні задовольняти
* Тестування вимог виконується широким колом фахівців

Література.

1. Соммервил И. Инженерия программного обеспечения / И.Соммервил.–М.: Издательский дом «Вильямс», 2002. –623 с.
2. Вигерс Карл. Разработка требований к программному обеспечению.Пер, с англ. - М.:Издательско-торговый дом «Русская Редакция», 2004. - 576с.
3. Лешек А. Мацяшек Анализ требований и проектирование систем. Разработка информационных систем с использованием UML.: Пер. с англ. - М.: Издательский дом "Вильямс". - 2002. - 432 с.